Transition Semantics

Mattox Beckman

beckman@iit.edu

Illinois Institute of Technology

Previous semantics both used proof trees:

- *Type derivations* specified the type of an expression.
- *Natural semantics* specified the value of an expression.

Today's semantics will use *transitions* to specify the value of an expression. By the end of lecture, you should know how to use transitional semantics to

- determine the value of an expression (i.e., be able to read)
- specify the meaning of a language (i.e., be able to write).

You should also know the Church-Rosser property and be able to give examples of languages that have it and languages that don't have it.

- There are many ways we can specify the meaning of an expression. One way is to specify the steps that the computer will take during an evaluation.
- An *evaluation* has the following form:

$$e_1 \longrightarrow e_2$$

where *e* is some expression, and e_2 is another expression, possibly a value.
Examples:

  - `if true then 4 else 38` $\longrightarrow$ `4`
  - `13 + 4 * 5` $\longrightarrow$ `13 + 20`

- Note well: $\longrightarrow$ indicates *exactly one* step of evaluation.

- In transition semantics we need to be able to distinguish between *values* and *expressions*.
  - A *value* is a valid *expression* that can not be evaluated any further.
  - (Note, the converse is not true.)
- Use letters U, V, and W to represent values.
- Use letters M, N, and L to represent expressions.

Here are three semantic rules for the `if` statement.

- `if true then` M `else` $N \to M$
- `if false then` M `else` $N \to N$
- $$\frac{L \to L'}{\texttt{if } L \texttt{ then } M \texttt{ else } N \ \to\ \texttt{if } L' \texttt{ then } M \texttt{ else } N}$$

In English:

- If the conditional part is `true`, evaluate the first branch.
- If the conditional part is `false`, evaluate the second branch.
- Otherwise, if the conditional part is not yet evaluated, evaluate it one step.

- These rules are for the short-circuit version of the boolean operators.

and

$$\texttt{true} \;\texttt{\&\&}\; N \to N$$
$$\texttt{false} \;\texttt{\&\&}\; N \to \texttt{false}$$

$$\frac{M \to M'}{M \;\texttt{\&\&}\; N \;\to\; M' \;\texttt{\&\&}\; N}$$

or

$$\texttt{true} \;\texttt{||}\; N \to \texttt{true}$$
$$\texttt{false} \;\texttt{||}\; N \to N$$

$$\frac{M \to M'}{M \;\texttt{||}\; N \;\to\; M' \;\texttt{||}\; N}$$

- These versions are for the “long” versions.

and

$$\frac{M \to M'}{M \texttt{ \&\& } N \;\to\; M' \texttt{ \&\& } N}$$

$$\frac{N \to N'}{U \texttt{ \&\& } N \;\to\; U \texttt{ \&\& } N'}$$

$$\texttt{true \&\& } V \to V$$
$$\texttt{false \&\& } V \to \texttt{false}$$

or

$$\frac{M \to M'}{M \texttt{ || } N \;\to\; M' \texttt{ || } N}$$

$$\frac{N \to N'}{U \texttt{ || } N \;\to\; U \texttt{ || } N'}$$

$$\texttt{false || } V \to V$$
$$\texttt{true || } V \to \texttt{true}$$

- These rules are boring. But we need to include them anyway.

$$\frac{M \rightarrow M'}{M \oplus N \rightarrow M' \oplus N} \qquad \frac{N \rightarrow N'}{V \oplus N \rightarrow V \oplus N'}$$

  Where $\oplus$ is +, -, >, <, . . .

- These rules are so boring that we don't include them.
  $0 + 0 \rightarrow 0 \quad 0 + 1 \rightarrow 1 \quad \ldots$
  $1 + 0 \rightarrow 1 \quad 1 + 1 \rightarrow 2 \quad \ldots$
  et cetera. . .

Evaluate: `if 3 > 2 then 5 + 9 else 2 * 4`

```
    if 3 > 2 then 5 + 9 else 2 * 4
→   if true then 5 + 9 else 2 * 4
→   5 + 9
→   14
```

Another common notation:
$\rightarrow^*$ means “zero or more transitions”, so for example `3` $\rightarrow^*$ `3`, and `if 3 > 2 then 5 + 9 else 2 * 4` $\rightarrow^*$ `14`

Choice 1 — Call By Value

$$\frac{N \rightarrow N'}{\texttt{let x} = N \texttt{ in } M \rightarrow \texttt{let x} = N' \texttt{ in } M}$$

$$\texttt{let x} = V \texttt{ in } M \rightarrow [V/\texttt{x}]\, M$$

$$(\texttt{fun x}{->}M)\ V \rightarrow [V/\texttt{x}]\, M$$

$$\frac{N \rightarrow N'}{(\texttt{fun x}{->}M)\ N \rightarrow (\texttt{fun x}{->}M)\ N'}$$

Choice 2 — Call By Name

$$\texttt{let x} = N \texttt{ in } M \rightarrow [N/\texttt{x}]\, M$$

$$(\texttt{fun x}{->}M)\ N \rightarrow [N/\texttt{x}]\, M$$

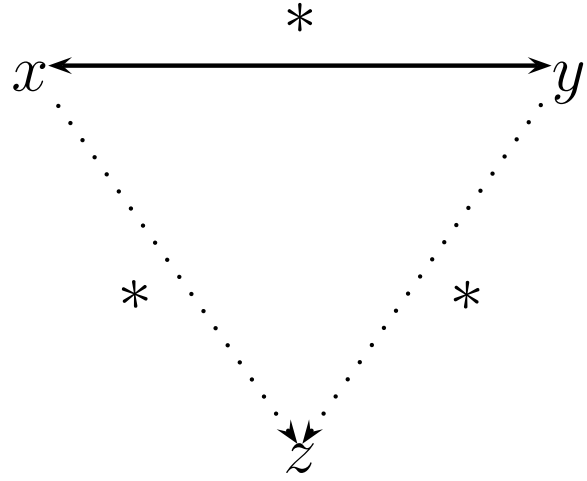
Evaluate: `let x = 2 + 3 in let y = x * x in x + y`

```
    let x = 2 + 3 in let y = x * x in x + y
→   let x = 5 in let y = x * x in x + y
→   let y = 5 * 5 in 5 + y
→   let y = 25 in 5 + y
→   5 + 25
→   30
```

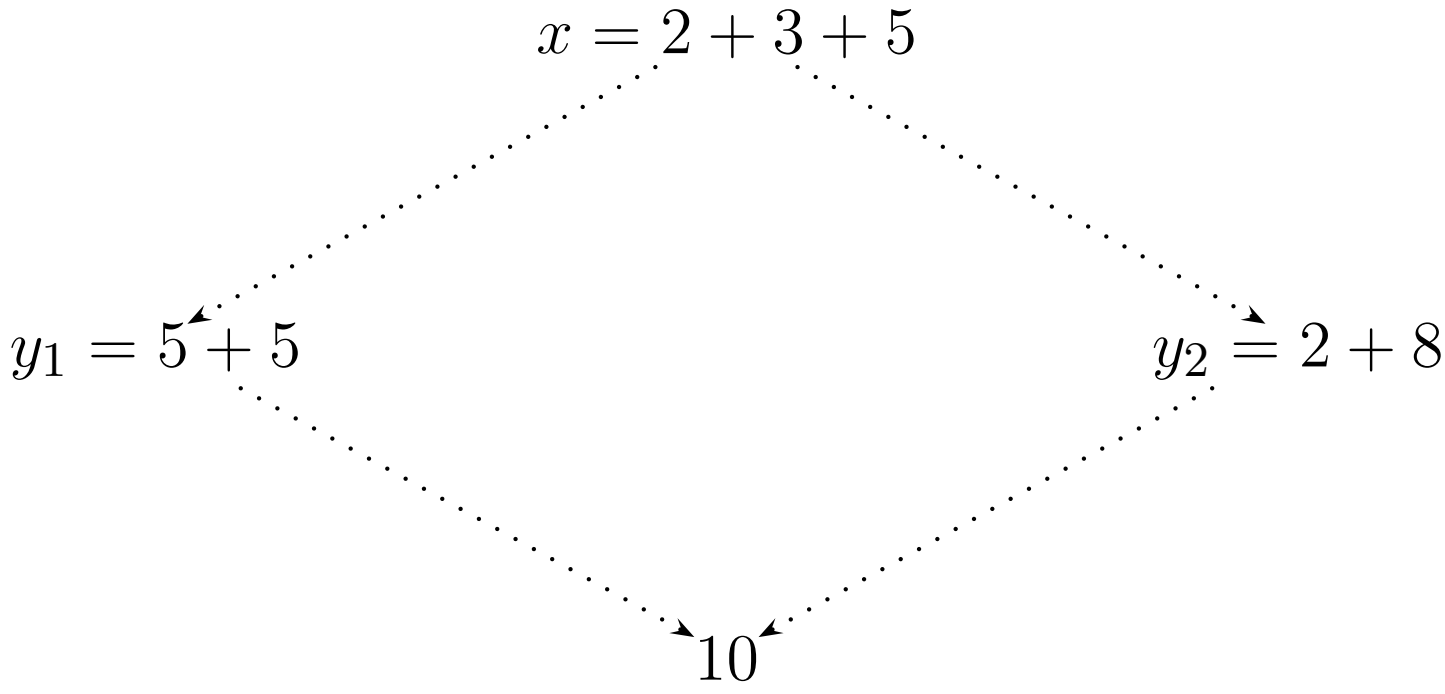
Transition semantics can be thought of as a *term-rewriting system*. Common questions:

- Does an expression always terminate?
- Can we tell if two expressions are equal?

Church-Rosser Property: If $x \rightarrow^* y$ and $y \rightarrow^* x$ then x and y normalize to the same value.



Confluence: If $x \longrightarrow y_1$ and $x \longrightarrow y_2$ then y_1 and y_2 normalize to the same value. (Confluence and the Church-Rosser Property coincide.)



This is also known as the “diamond property”

- Alonzo Church and J. Barkley Rosser proved that the λ-calculus has these properties in 1936.
- Very important for theorem provers.
- Most programming languages have this property... some of the time...
- One Benefit: you can check for equality of x and y by evaluating them.

Not Church-Rosser

```
1 int i = 10;
2 int sum(int x, int y) { return x + y; }
3
4 cout << sum(i++, i*=2); // 11 + 22 or 21 + 20?
```

Problems

1. Use transition semantics to calculate the value of
   `let f = fun x -> x * x in f 5`
2. Suppose we add a `match/with` type statement to λ-calculus. Will it still have the Church-Rosser property?
3. Suppose we specify the order of evaluations for arguments in C++. Does that restore the Church-Rosser property?

1. Evaluate: `let f = fun x -> x * x in f 5`

```
    let f = fun x -> x * x in f 5
→   (fun x -> x * x) 5
→   5 * 5
→   25
```

2. Suppose we add a `match/with` type statement to λ-calculus. Will it still have the Church-Rosser property? **Yes**

3. Suppose we specify the order of evaluations for arguments in C++. Does that restore the Church-Rosser property? **No**

```
1 char *i = new char('x');
2 cout << (int) i;   // different each time
```